CLAIMS

1. A method of replacing an implementation module which is being accessed through an interface module by a plurality of threads from an application, including the steps of:
   - i) creating a plurality of private variables corresponding to the plurality of threads;
   - ii) setting a replace module variable;
   - iii) when the replace module variable is set:
     - a. blocking threads from entering the implementation module; and
     - b. when all the private variables are in a reset state, replacing the implementation module;

   wherein the private variable is never in a reset state when the thread is within the implementation module, wherein the use of locks within the performance path of the interface module is not required, and wherein threads and corresponding private variables are created and destroyed dynamically.

2. A method as claimed in claim 1 wherein the implementation module is a non-recursive module.

3. A method as claimed in claim 1 wherein the implementation module is a recursive module.

4. A method as claimed in claim 3 including the step of:

   creating a plurality of counter variables corresponding to the plurality of threads;

   wherein each counter is incremented when the corresponding thread enters the implementation module and decremented when the thread exits the implementation module.

5. A method as claimed in claim 4 wherein the private variable is in a set state when the value of the counter is above zero and the private variable is in a reset state when the value of the counter is zero or below.

6. A method as claimed in claim 5 wherein each counter is only readable and writable by its corresponding thread.

7. A method as claimed in claim 1 wherein step (iii) is performed by the interface module.

8. A method as claimed in claim 7 wherein each private variable is modifiable by that variable's corresponding thread.

9. A method as claimed in claim 8 wherein each private variable is readable by all the threads.

10. A method as claimed in claim 9 wherein the implementation module is a library.

11. A method as claimed in claim 1 wherein the private variables and the replace module variable are defined as cache coherent.

12. A method as claimed in claim 1 wherein a thread performs the step (iii).

13. A method as claimed in claim 12 wherein a mutual exclusion primitive is used within step (iii) to ensure that only a single thread performs steps (a) and (b).

14. A method as claimed in claim 1 wherein checking of flags within an array is not required in the performance path of the interface module.

15. A method of synchronizing a plurality of threads for the performance of an action which affects a resource accessed within a portion of code, including the steps of:

i) registering for each thread a corresponding private variable;

ii) each thread setting the private variable when that thread enters the portion of code;

iii) setting a perform action variable when the action is to be performed;

iv) when a thread is within the portion of code and the perform action variable is set, the thread:

a. resetting the private variable;

b. when the private variables for all threads are not set:

i. performing the action; and

ii. resetting the perform action variable;

c. setting the private variable; and

v) when a thread is within the portion of code and the perform action variable is reset, the thread:

d. using the resource; and

e. resetting the private variable;

wherein the threads may be dynamically created and destroyed.

16. A method as claimed in claim 15 wherein the use of locks within the performance path of the portion of code is not required.

17. A method as claimed in claim 16 including the step of:

when the thread has used the resource in step (v) and the perform action variable is set, the thread performing step (b).

18. A method as claimed in claim 16 wherein the private variables and the perform action variable are cache coherent.

19. A method as claimed in claim 16 wherein when a thread is performing the action in step (b), all other threads are blocked from performing step (b).

20. A method as claimed in claim 19 wherein the other threads are blocked by use of a mutual exclusion primitive.

21. A method as claimed in claim 16 wherein each thread registers their corresponding private variable.

22. A method as claimed in claim 21 wherein the registration of each private variable occurs when the corresponding thread is created.

23. A method as claimed in claim 21 wherein the registration of each private variable occurs when the corresponding thread enters the interface module for the first time.

24. A method as claimed in claim 16 wherein a private variable is deregistered when its corresponding thread is destroyed.

25. A method as claimed in claim 16 wherein the resource is the kernel or operating system of a machine upon which a process containing the threads is operating and the action is the migration of the process to a new machine.

26. A method as claimed in claim 16 wherein the resource is an implementation module, the portion of code is an interface module for an implementation module and the action is the replacement of the implementation module.

27. An interface module for an implementation module, including:
   i) a plurality of private variables for correspondence to a plurality of threads, each private variable to be readable by all threads and writable only by the corresponding thread, and each private variable arranged to be in a SET state or a RESET state;
   ii) a replace module variable; and
   iii) program code arranged for registering the privates variables for created threads, deregistering the private variables for destroyed threads, blocking threads from entering the implementation module, and replacing the implementation module when all the registered private variables are in a RESET state;

   wherein the use of locks within the performance path of the interface module is not required and the threads are dynamically created and destroyed.

28. A system for replacing an implementation module, including:
   i) a memory which stores a plurality of private variables corresponding to a plurality of threads;
   ii) a memory which stores a replace module variable; and

iii) a processor arranged for registered the private variables for created threads, deregistering privates variables for destroyed threads, setting and resetting the registered private variables when instructed by the corresponding thread, setting the replace module variable, and when the replace module variable is set:

a. blocking entry of threads to the implementation module; and

b. when all the registered private variables are in a reset state, replacing the implementation module;

wherein the threads are dynamically created and destroyed.

29. A system as claimed in claim 28 wherein the processor is further arranged for resetting the replace module variable when the implementation module has been replaced.

30. A system as claimed in claim 29 wherein the processor is further arranged for unblocking the threads when the replace module variable has been reset.

31. A computer system for performing the method of claim 1.

32. A computer system for performing the method of claim 15.

33. Software for performing the method of claim 1.

34. Software for performing the method of claim 15.

35. Storage media containing software as claimed in claim 33.

36. Storage media containing software as claimed in claim 34.